QMine: A Framework for Mining Quantitative Regular Expressions from System Traces

Pradeep Kumar Mahato, Apurva Narayan
Department of Computer Science
The University of British Columbia
BC, Canada

Motivation

Data is everywhere !!

Modern-day software systems are complex and produce an exorbitant amount of data. Understanding system behaviour using these data is crucial.

Common datasources:

- Medical instruments
- Geolocation devices
- Heavy machinery and their controllers

Fig: Sample heartbeat record

Source: Pixabay, https://www.pexels.com/photo/blur-chart-check-up-curve-415779/

Motivation

Temporal properties provide information about the occurrence of events. It draws insights from the system specification.

Challenges

- Extracting such properties is challenging
- Specification are sometimes loosely specified
- Lack of proper specifications

Fig: Stock market dashboard sample

Source: Pixabay, https://www.pexels.com/photo/blue-and-yellow-graph-on-stock-market-monitor-159888/

Objective

Mine for temporal properties in the form of quantitative regular expressions from system traces.

Few major contribution of our work

- Present novel framework to mine temporal properties using Quantitative Regular Expressions
- Justify algorithmic characterization with space and time
- Validate using industrial system traces

Outline

- Background
- Methodology
- Experiments
- Conclusion
- Future Work

Background

Event, traces and their representation

Event

The alphabet of events is a finite alphabet of strings

Trace

Group of events forms a trace

```
[ 43778.198] (--) NVIDIA(GPU-0): DFP-0: disconnected
[ 43778.198] (--) NVIDIA(GPU-0): DFP-0: Internal TMDS
[ 43778.198] (--) NVIDIA(GPU-0): DFP-0: 165.0 MHz maximum pixel clock
[ 43778.198] (--) NVIDIA(GPU-0):
```

Fig: Sample log from Xorg in ubuntu

Notations

$$\text{Events} : \alpha_i$$
$$\text{Global alphabet set} : \Sigma$$
$$\text{Trace contains } \langle \alpha 1,\ \alpha 2, \cdots,\ \alpha_n \rangle \in \Sigma$$

Quantitative Regular Expressions

Proposed by Rajeev Alur et al.[1], Quantitative Regular Expressions (QRE) are representations to evaluate quantitative values using regular expressions defined over an input domain **D** and cost domain **C**

QRE expression for performing average :

$$a = atom(x \to x.type = M, x \to x.val)$$

$$b = atom(x \to x.type = M, x \to 1)$$

$$num = iter(a, a, (x, y) \to x + y)$$

$$den = iter(b, b, (x, y) \to x + y)$$

$$avg : QRE\langle X, Y \rangle = combine(num, den, (x, y) \to x/y)$$

X and **Y** are input and output data types respectively

Quantitative Regular Expression Template (QRET)

Definition 4 (QRET). *A QRET is a valid template if it contains at least two events* $(\alpha_i, \alpha_j) \in \Sigma$ *and a series of* m *quantitative values* $q_1 \cdots q_m$ *bounded by* α_i *and* α_j

More precisely, the most rudimentary form of *QRET* can be explained as QRET = $\alpha_1 \mathbb{R} \alpha_2$, where $\mathbb{R}$ is a set of real numbers.

Sample

QRET : $0\ M\ 1$

$Event_0$ followed by a quantitative value, M, which is then followed by $Event_1$

Placeholder 0 and 1 are replaced by all events in the alphabets $\langle \alpha_0,\ \alpha_1, \cdots \rangle\ \in \Sigma$

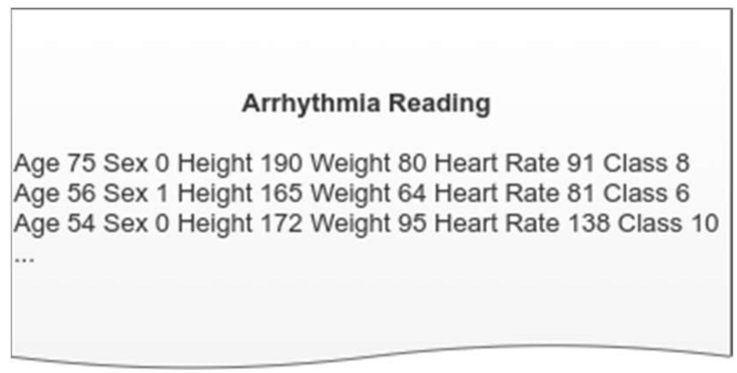
QTrace : A trace complaint of QRET instance

A series of events with quantitative values between them forms a **QTrace** file.

Formally, each quantitative value ($\mathbb{R}$) is enclosed with starting event α_i and closing event α_{i+1}

<u>Sample</u>

α_a 123 α_b : Quantitative value 123 bounded by α_a and α_b



Arrhythmia Reading

Age 75 Sex 0 Height 190 Weight 80 Heart Rate 91 Class 8
Age 56 Sex 1 Height 165 Weight 64 Heart Rate 81 Class 6
Age 54 Sex 0 Height 172 Weight 95 Heart Rate 138 Class 10
...

Fig: Sample illustration of Arrhythmia Dataset

Methodology

QMine Framework

QMine Core uses parallelization for mining

Results are stored in **Mine Map** for future query.

This avoids re-mining

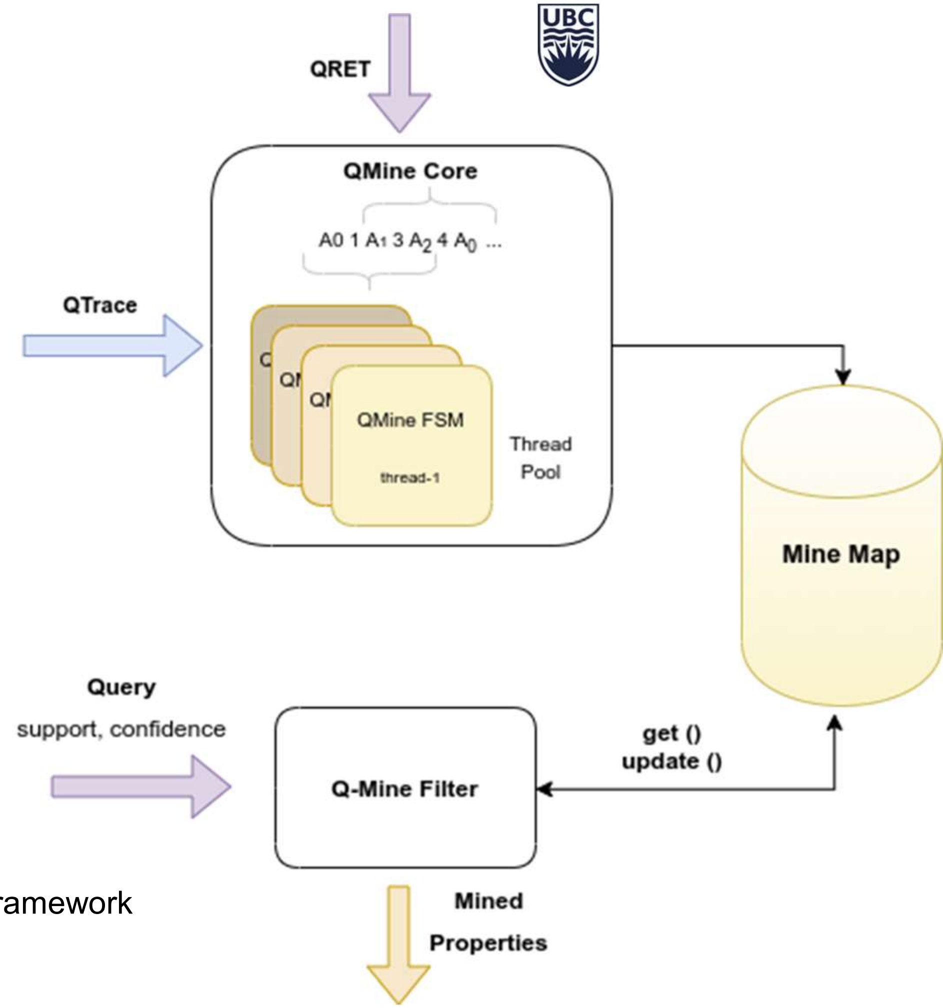


Fig: QMine framework

QMine Algorithm

Algorithm 1: QMine(QRET, QTrace, Σ, ξ)

Input: QRET, QTrace, alphabet Σ, mine-map ξ
Result: Mined properties for QTrace

Generate all permutations of *QRET* from Σ
Initialize Finite State Acceptor (FSA)
Set all threads with ξ and the FSA
Divide QTrace segments per thread
foreach *segment* $\in$ *QTrace segments* **do**
    mineInternal (segment, ξ, FSA)
end

Fig: QMine algorithm

Algorithm 2: mineInternal(segment, ξ, FSA)

Input: QTrace Segment, ξ, FSA
Result: Mined property satisfied by a QTrace segment

action **EVENT**
    record patterns for QTrace Segment match
action **NUM**
    push digit to ξ
action **FINISH**
    match overall pattern
    push ξ into global shared space & increment count
    return TRUE `// Accepting state`
action **ERROR**
    remove inserted values from ξ
    return FALSE `// Error State`

QMine Threading

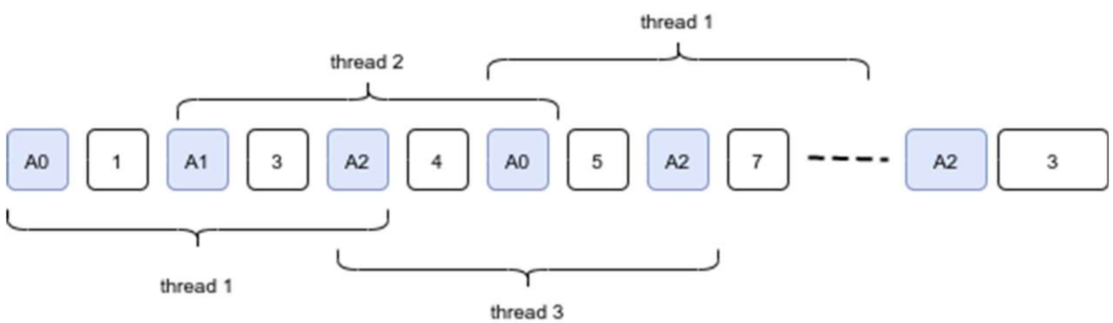


Fig: Thread workload distribution

Experiments

Heartbeat analysis for arrhythmia

Dataset Arrhythmia patients
Source UCI Machine Learning Repository[2]

Patients 452
Classes 15

Class 1	Normal reading
Class 2	Patient with coronary artery diseases
Class 3 - 15	Different groups for arrhythmia patients

Events (in consideration) Age, Sex, Height, Weight, Heart Rate, and Class

Objective Identify patterns for arrhythmia patients

QRET 0 M 1 M 2 M 3 M 4 M 5 M 6

Alphabet Set Size 7

Mining Combination 7 ! i.e 5040 total patterns

Heartbeat analysis for arrhythmia

Age	Sex	Count	Height (cm)		Weight (kg)		Heart Rate (bpm)		Class
	0 -Male 1-Female		Mean	Variance	Mean	Variance	Mean	Variance	Class
0-39	0	36	174.194	24.8789	77.8333	124.806	92.0833	69.6319	1
	1	118	159.356	14.3987	60.8305	131.175	81.4661	52.5031	
	0	5	167.2	5.36	58	49.2	98.8	21.36	2
	1	12	161.333	65.2222	62.1667	270.472	90.5	63.5833	
	0	13	168.154	6.74556	69.3846	282.544	95.8462	19.2071	Not 1
	1	28	162.5	42.25	61.6429	244.944	83.2857	108.347	
40-99	0	123	171.407	42.5665	76.0569	187.289	89.9024	60.771	1
	1	183	160.541	25.9532	68.0383	188.933	80.9071	55.6253	
	0	29	169.103	40.9893	80.8276	746.212	94.8621	96.3948	2
	1	40	158.7	20.61	73.7	172.81	84.45	66.4475	
	0	115	170.13	43.7656	76.1043	249.641	96.8174	374.81	Not 1
	1	77	157.987	21.2596	71.2208	144.432	90.6883	573.955	

Table: Summary of quantitative values from arrhythmia dataset

Synthetic analysis (stress testing)

Dataset Self generated adhering to QRET pattern

CPU : Intel i7-2630 QM (2.6 Ghz max boost)

Max Threads 16

FSM framework Ragel [3]

Events (in consideration) < variable >

QRET < variable >

Alphabet Set Size < variable >

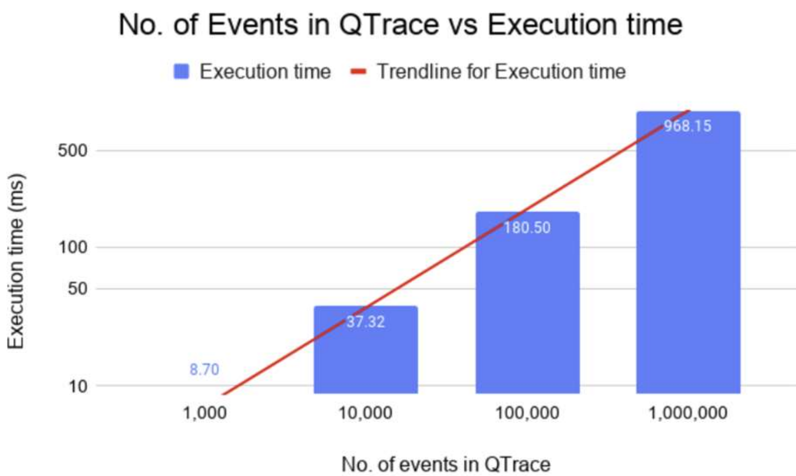
Mining Combination < variable >

Synthetic analysis (stress testing)

QRET	Alphabet Size	Total QRE Instances	Compile Time Taken (ms)
0M1	10	90	2,840
	50	2450	2,955
	500	249500	61,274
	1000	999000	247,728
0M1M2	10	720	2,912
	20	6840	3,258
	50	117600	23,114
	100	970200	638,822

Table: Analysis with QRET pattern, alphabet size and compilation time

Synthetic analysis (stress testing)



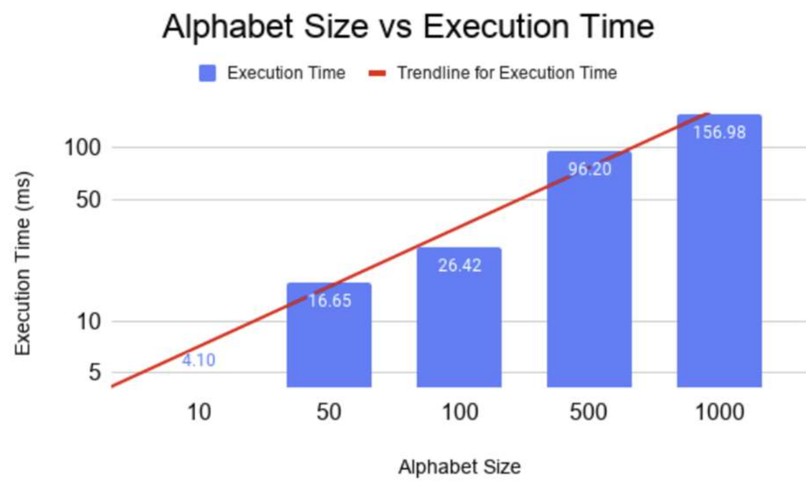


Table: Analysis with varying alphabet size and event placeholders

Algorithmic complexity

Complexity Analysis	
Time	$\mathrm{O}(\Sigma^{\tau} + L)$
Space	$\mathrm{O}(\Sigma^{\tau} + L)$
Execution	$\mathrm{O}(\Sigma^{\tau} + L/(\texttt{th-count}) + th_{oh})$

Table: Complexity analysis *

* placeholder length denoted by **τ**

Conclusion

- We presented a novel QMine framework for extracting and inspecting for interesting patterns
- Scalability of the framework is almost linear*
- Our algorithm is robust, sound and complete

Few of the future works could be:

- Improve mining with hard constraints
- Dynamic pattern inspection based on system behaviour

* Once the FSM model is generated, our mining is almost linear via parallelization

Source: Pixabay, https://pixabay.com/illustrations/white-male-3d-model-isolated-3d-2064872/

References

1. R. Alur, K. Mamouras, and C. Stanford, "Modular quantitative monitoring," Proceedings of the ACM on Programming Languages, vol. 3, no. POPL, pp. 1–31, 2019.
2. C. Blake and C. Merz, "UCI machine learning repository," 1998. [Online]. Available: http://www.ics.uci.edu/mlearn/MLRepository.html
3. A. Narayan, G. Cutulenco, Y. Joshi, and S. Fischmeister, "Mining timed regular specifications from system traces," ACM Transactions on Embedded Computing Systems (TECS), vol. 17, no. 2, pp. 1–21, 2018
4. A. Thurston, "Ragel state machine compiler," 2015
5. C. Lemieux, D. Park, and I. Beschastnikh, "General LTL Specification Mining," in Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on. New York, NY, USA: ACM, 2015, pp. 81–92.
6. R. Alur, D. Fisman, and M. Raghothaman, "Regular programming for quantitative properties of data streams," in European Symposium on Programming. Springer, 2016, pp. 15–40

\- Fin -

Thank you